To appear in the 14th International Object-oriented and Entity-Relationship Modelling conference, Gold Coast, 1995 (to be published in LNCS)

Integrating and Supporting Entity Relationship and Object Role Models

John R. Venable and John C. Grundy

Department of Computer Science, University of Waikato
Private Bag 3105, Hamilton, New Zealand
email: jvenable@cs.waikato.ac.nz or jgrundy@cs.waikato.ac.nz

Abstract. This paper describes the conceptual integration and computer-based support of two important groups of conceptual data models, Entity Relationship Models and Object Role Models (e.g. NIAM). We perform conceptual integration using the conceptual data modelling language CoCoA to specify separate data models of individual notations. We then merge these into an integrated conceptual data model for both notations. These data models form the basis of the repository for an I-CASE tool supporting modelling with both notations, with full consistency management between the two notation data models.

1 Introduction

1.1 ER and NIAM Models

Conceptual data modelling is an important perspective used in describing information systems during requirements analysis and specification. Conceptual data models are information systems modelling languages (ISMLs) that are used to describe, reason about, or document the logical structure and meaning of data and the concepts they represent. They are not concerned with what the data is used for or how. They are also unconcerned with how data is represented within a computer-based information system.

Two important groups or classes of conceptual data models are Entity Relationship (ER) Models and Object Role Models (ORM). Each group has its advantages and disadvantages as well as its adherents and critics. The ER model group follows from the work of Chen [2], and ER models have become very popular for Information Systems development. ER models utilise attributed entities and relationships to describe information structure. Fig. 1 shows an example of an ER data model for a simple invoicing system.

ORM models follow from the work of Nijssen and others at Control Data in the Netherlands. The most well known ORM is NIAM, as described in [15] and refined in [9]. ORM offer a well-thought-out system of constraints, rigorous means of dealing with higher arity relationships (i.e., n-ary with n>2), and an effective means of communicating with users via examples of data. Fig. 1 also shows an example of a NIAM data model for a simple invoicing system.

Some problems suit ER modelling while others suit ORM modelling, and developers have a preference to which notation they use. In order to facilitate system development

using both of these modelling techniques simultaneously, an I-CASE tool is required which supports integrated modelling with both notations. Consistency management between ER and NIAM models must be supported in order to effectively use both notations on the same problem domain. Without such tool support, effective utilisation of both models is difficult, as inconsistencies are difficult to manually detect and correct.

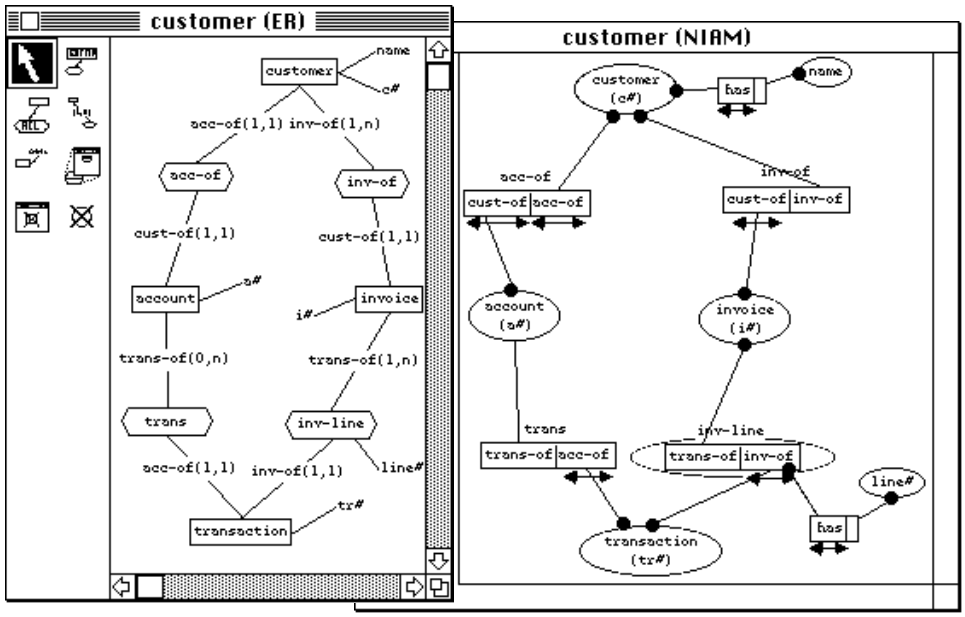

Fig. 1. ER and NIAM data models for an invoicing Information System.

1.2 Related Research

Integrated ISEEs (or Integrated CASE tools and programming environments) allow designers to analyse, design, and implement Information Systems from within one environment, providing a consistent user interface and consistent repository (data dictionary). They help to minimise inconsistencies that can arise when using several separate tools for information systems development [16, 12].

Some work has been done on the static integration of notations. We base this work on Venable [14] who has performed detailed analyses and integrations of both data flow models and conceptual data models. Campbell and Halpin [1] have also analysed abstraction techniques for conceptual schemas, including those in the ER and NIAM models. Falkenberg and Oei [3] have proposed a metamodel hierarchy but it has not yet been applied to the ER and ORM areas. Data modelling has been used to compare different notations [10] and support methodology engineering [7]. Process-modelling has also been applied to compare and integrate notations [13]. Little has been done to translate conceptually integrated notations into tool-based implementations.

Limited dynamic notation integration is supported by many CASE tools, such as Software thru Pictures™ [16]. These ICASE environments allow developers to analyse

and design software using a variety of different notations, with limited inter-notation consistency. Such tools do not generally support complex mappings between the design notations, such as propagating an ER relationship addition to a corresponding OOA/D or NIAM diagram. The implementation of these environments is generally not sufficient to allow different design notations to be effectively integrated, and consistency between design and implementation code is often not maintained [8]. FIELD [12] and Dora [11] provide abstractions for keeping multiple tools and textual and graphical views consistent under change. They do not, however, provide any mechanism for propagating changes between views which can not be directly applied by the environment, such as ER relationship changes to NIAM or OOA/D relationship changes. Thus changes which can not be automatically translated to another notation are not supported.

1.3 Our Approach

To produce a true I-CASE tool for ER and NIAM modelling, we utilise the approach described in [5]. First, conceptual data models are developed for both the ER and NIAM notations. Second, an integrated conceptual data model is derived which captures the concepts of both the ER and NIAM. Third, dynamic data mappings between the individual models are developed which describe how changes to data in one notation can be reflected as data changes in the other. Forth, individual CASE tools are developed for the ER and NIAM notations, using their individual conceptual data models to specify the tool repositories. Finally, these individual modelling tools are integrated into a single I-CASE tool by defining an integrated repository based on the integrated conceptual data model. The individual repositories are kept consistent by using the data mappings to specify how data changes in one repository are propagated to the integrated repository and then onto the other notation's repository.

2 Integration of the Notation Conceptual Data Models

2.1 CoCoA

We use the CoCoA conceptual data modelling language [14] as a meta-model for modelling Information System Modelling Languages (ISMLs) and their concepts. CoCoA is designed to support modelling of complex problem domains and extends existing Entity Relationship (ER) models. Fig. 2 depicts the seven main CoCoA abstractions. Entities are the things in a problem domain and attributes describe and/or identify them (Fig. 2 (a)). Named relationships have the semantics of ER relationships, and are composed of named roles, played by entities. Cardinalities are indicated with each role (Fig. 2 (b)).

CoCoA supports generalization and specialization, and where specialization is based on a partitioning attribute, that attribute is shown (Fig. 2 (c)). CoCoA extends other ER models by the implicit use of categories, allowing more than one entity (type) to play the same role in the same named relationship (Fig. 2 (d)). CoCoA derives its name from a fifth data modelling concept, that of Complex Covering Aggregation. Covering aggregation distinguishes the aggregation of entities into composite entities from the aggregation of attributes into entities. Complex covering aggregation is distinguished from simple covering aggregation in that aggregation of named relationships into the composite entity is allowed (Fig. 2 (e)). CoCoA supports aliases, which are useful for model integration, showing old local names together with standardized names for synonyms (Fig. 2 (f)). Derived concepts (attributes, entities, named relationships, or

covering aggregation relationships) are annotated with a '*' (Fig. 2 (g)).

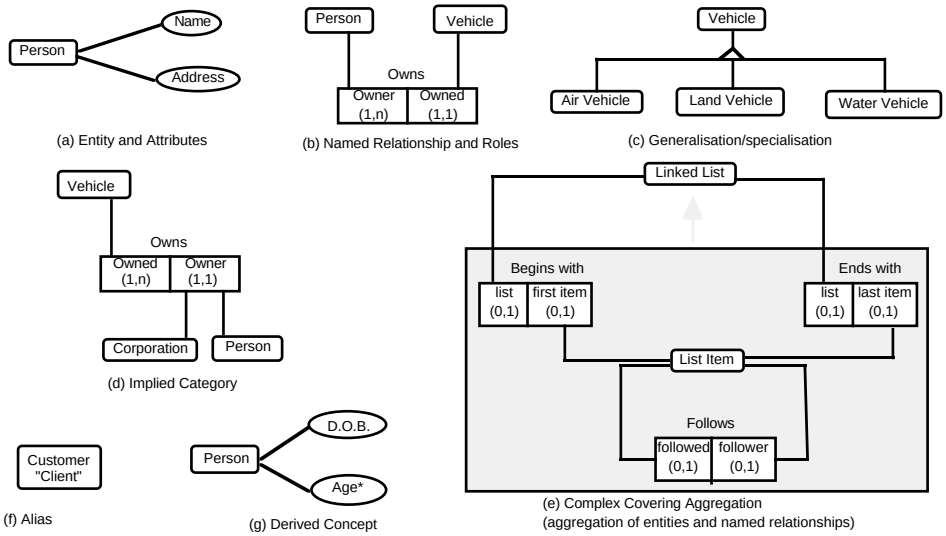

Fig. 2. The CoCoA model notation

2.2 Individual Notation Conceptual Data Models

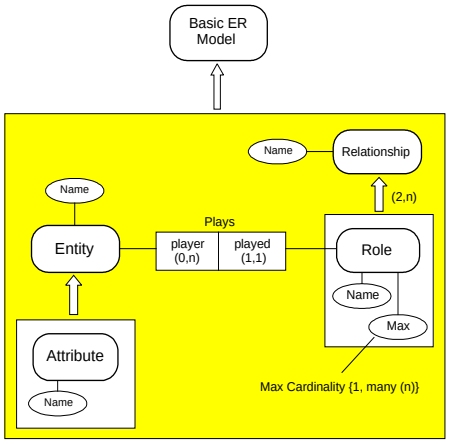

Fig. 3. ER conceptual data model.

We have used CoCoA to derive conceptual data models for ER and NIAM. The data model describing the fundamental abstractions of ER models is shown in Fig. 3. Entities are named and have zero or more named attributes. Relationships are named and have two or more named roles. Roles link entities and relationships and may include a maximum cardinality. Extensions to this basic ER schema include provision for entity subtyping, optional and mandatory roles, and distinguished key attributes of entities [14].

Fig. 4 shows NIAM's main abstractions. A NIAM entity is named and may have a

reference, made by one or more named labels. Fact types are named and have one or more roles. The “derived” attribute of the fact type entity is marked as derived (by the asterisk) because it’s value is true if it is related to a derivation rule. Roles link entities to facts, and are named. Nested fact types are both entities and facts i.e. have roles but also behave as entities, being linked to zero or more facts via further roles. A CoCoA model of other NIAM constraints is omitted for brevity, but can be found in [14]. NIAM derivation rules are not specified further because they are not fully specified by Nijssen and Halpin [9].

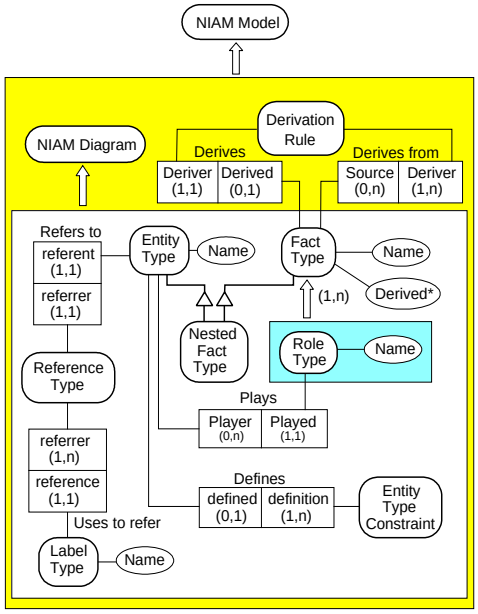

Fig. 4. NIAM entities conceptual data model.

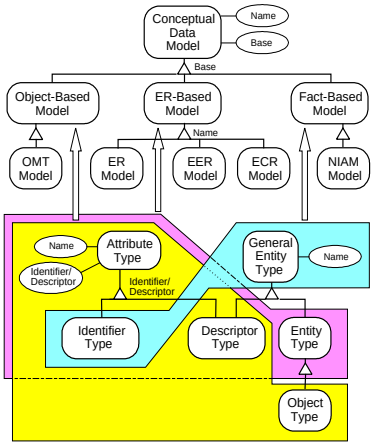

Fig. 5. An integrated conceptual data model.

2.3 An Integrated Conceptual Data Model

We have developed an integrated data model which captures the overlaps between ER and NIAM. We have included in this data model the OMT notation, an object-oriented modelling approach, as shown in Fig. 5. The ER and OMT models differentiate between entities and attributes, whereas NIAM integrates these concepts into a general entity type. The main difference between the OMT and ER conceptual data models is OMT's support for class methods. The overlaps between the notations are indicated by covering aggregation showing the composition of each data model from the integrated data model entities and relationships. Further discussion of relationship type classifications is in [14].

3 Internotation Mappings

Our integrated conceptual data model specifies the *static* integration of the two notations i.e. which ER concepts correspond to which NIAM concepts and vice-versa. This integrated data model can form the specification for an integrated tool repository for ER and NIAM notations. It does not, however, specify *dynamic* mappings between the two notations i.e. when an ER model is changed, what is the affect on the NIAM model for the same problem? It is straightforward to translate an ER change into the integrated repository change, as the integrated model fully describes the ER data model. It is often more difficult to translate a change to the integrated data model into a change in the NIAM model, as some of the concepts modified may not directly correspond. This is also true in the reverse direction (NIAM to integrated model to ER model).

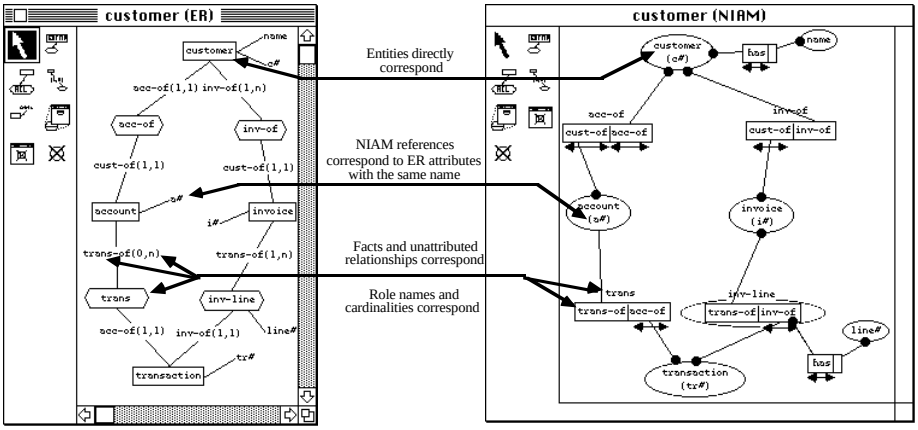

Fig. 6. Direct mappings between ER and NIAM models.

Some dynamic mappings are quite straightforward, and we term them *direct* mappings. For example, ER entities map directly to NIAM entities, so adding, updating or deleting an ER entity can be automatically reflected as appropriate changes on the corresponding NIAM entity. Unattributed ER relationships correspond to NIAM facts, and thus adding an ER relationship and roles can be directly propagated to the addition of a NIAM relationship and roles. Renaming ER roles and relationships can be translated into renaming NIAM roles and facts. NIAM entity references, which are used to refer to an entity, correspond to ER attributes of the same name. Direct mappings between the two

models are shown in Fig. 6.

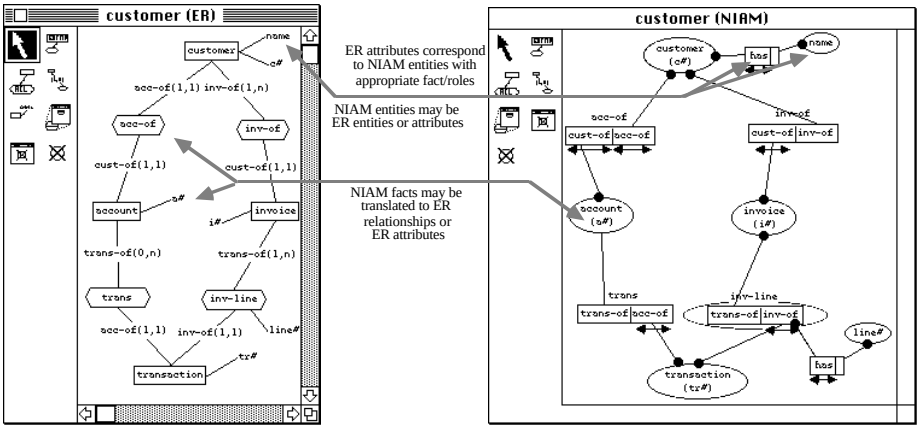

Fig. 7. Indirect mappings between ER and NIAM models.

Some concepts do not directly map to concepts in the other notation, and we term these *indirect* mappings. See Fig. 7. For example, ER attributes must be mapped to NIAM entities, with appropriate NIAM roles and a fact to link the "attribute" entity to its owner. In Fig. x, a designer adding a "name" attribute to the ER model results in a corresponding "name" entity and binary fact ("has") being added to the corresponding NIAM model. Similarly, if a designer adds a NIAM entity, this may be represented in the ER model by an entity or an attribute. Addition of a NIAM fact to an entity may translate to an ER relationship to the entity, or may translate to an ER attribute. Such situations require user intervention to determine what the correct translation to the ER model should be (see the following section).

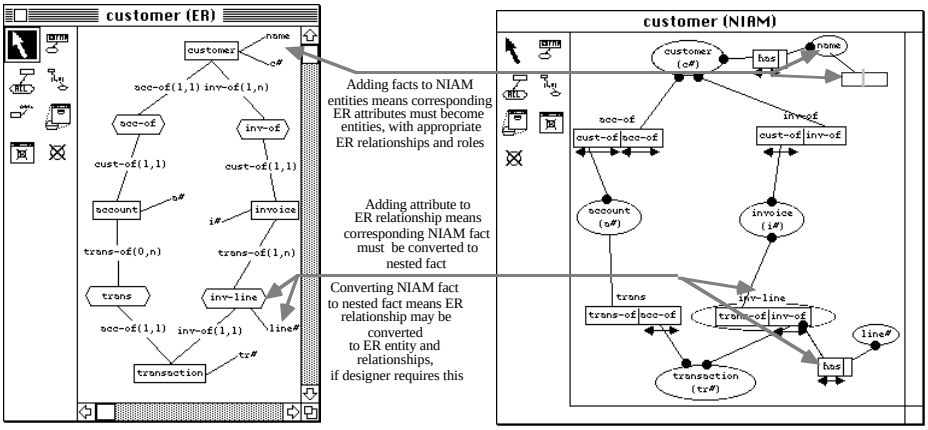

Fig. 8. Further indirect mappings between ER and NIAM models.

NIAM supports many constraints between facts, such as uniqueness values for single or

composite roles. Subtyping and mandatory/optional roles correspond to the same concepts in most ER models. The other constraints do not have similar concepts in ER models, and can not be translated into ER changes. However, designers should be informed when using ER diagrams that such untranslatable constraints have been added to the corresponding NIAM model.

Some mappings between the notations can be translated initially, but may need to be modified depending on further development of the system being designed. For example, if the customer name attribute was added to the ER model, a corresponding NIAM entity and fact would be added. If the designer subsequently added further facts about this entity to the NIAM view, such as first name and last name, the ER customer name attribute would need to be converted to an ER entity, with appropriate relationship and roles to the customer entity. Similarly, if attributes are added to an ER relationship, the corresponding NIAM fact must be converted to a nested fact type, which can have facts of its own. Thus some modifications to one notation result in many modifications to the other model. Some may be defaulted by an I-CASE tool, but some user intervention may be required. Fig. 8 illustrates complex indirect mappings.

4 An Integrated CASE Tool for NIAM/ER Modelling

4.1 MViews

NIAMER is implemented as a collection of classes, specialised from the MViews framework [4, 6]. MViews supports the construction of Integrated Software Development Environments (ISDEs) by providing a general model for defining software system data structures and tool views, with a flexible mechanism for propagating changes between software components, views and distinct software development tools.

MViews describes ISDE data as *components* with *attributes*, linked by a variety of *relationships*. Multiple views are supported by representing each view as a graph linked to the base software system graph structure. Each view is rendered and edited in either a graphical or textual form. Distinct environment tools can be interfaced at the view level (as editors), via external view translators, or multiple base layers may be connected via inter-view relationships.

When a software or view component is updated, a change description is generated. This is of the form UpdateKind(UpdatedComponent, ...UpdateKind-specific Values...). For example, an attribute update on Comp1 of attribute Name is represented as: update(`Comp1,Name,OldValue,NewValue`). All basic graph editing operations generate change descriptions and pass them to the propagation system. Change descriptions are propagated to all related components that are dependent upon the updated component's state. Dependents interpret these change descriptions and possibly modify their own state, producing further change descriptions. This change description mechanism supports a diverse range of software development environment facilities, including semantic attribute recalculation, multiple views of a component, flexible, bi-directional textual and graphical view consistency management, a generic undo/redo mechanism, and component "modification history" information.

New software components and editing tools are constructed by reusing abstractions

provided by an object-oriented framework. ISDE developers specialise MViews classes to define software components, views and editing tools to produce the new environment. A persistent object store is used to store component and view data.

4.2 NIAMER Architecture

NIAMER was built by integrating two independently developed CASE tools. MViewsER [5] provides multiple graphical ER views and textual relational schema views. MViewsNIAM provides multiple NIAM modelling views. These two tools were integrated by developing an integrated repository, based on the integrated conceptual data model from section 2. The items in this integrated repository are linked to items in the individual tool repositories. Changes to ER tool data are translated into changes to the integrated repository data and then into changes on NIAM tool data (and vice-versa), by the inter-repository relationships.

An advantage of using an integrated repository to maintain inter-notation consistency is when extending the environment. For example, the authors have developer another environment supporting integrated EER and OOA/D modelling, called OOEER [5], with many of the NIAM and ER mappings used in OOEER. Thus it is far easier to integrate OOEER and NIAMER by using the integrated repository, to produce an environment supporting EER, OOA/D and NIAM modelling, than by respecifying many of the mappings already implemented in each environment. In fact, a third integrated repository could be used to link the integrated NIAMER and OOEER repositories.

4.3 Examples of NIAMER Consistency Management

Some translations between NIAM and ER diagrams can be fully automated by NIAMER. In this case, when a designer selects a view from the other notation from which they have been working in, it is updated to reflect any changes which can be automatically carried out. Some updates can not be automatically carried out. In this case, NIAMER allows designers to browse a “modification history” for the view or the components rendered in the view, to determine if manual updates are needed to complete a translation.

Fig. 9 shows the modification history for the NIAM transaction entity. This contains a list of human-readable change descriptions which inform the designer of changes which have affected the transaction entity. The lines prefixed by ‘ER:’ have been carried on ER model diagrams, affecting the ER transaction entity. These have been sent to the NIAM transaction entity and are stored to document ER changes affecting this entity. NIAMER has also translated some of these updates into NIAM model updates, indicated by the following lines prefixed by a ‘→’. Changes not prefixed by a ‘→’ have been manually carried out by the designer to maintain full internotation consistency.

NIAMER supports textual views showing relational database tables for entities and facts/relationships. This facility is provided by MViewsER, but as the NIAM model is kept consistent with the ER model, relational tables can be produced for the NIAM model as well. These textual views can be kept consistent with the graphical views by expanding change descriptions into the view’s text. Some expanded changes can be automatically applied to the textual view on programmer request. For example, update 12 in Fig.9 can be automatically applied by NIAMER, resulting in the acc-of table field being renamed.

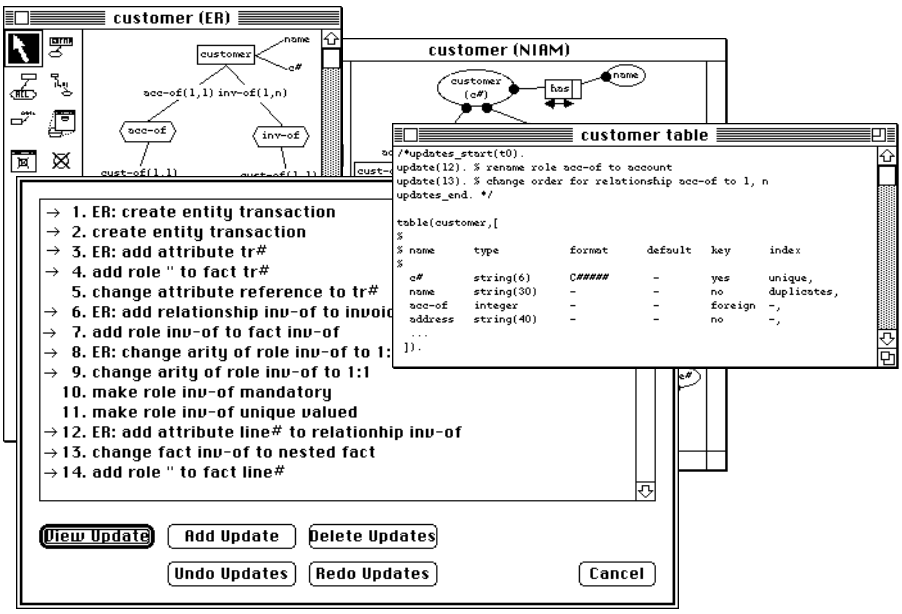

Fig. 9. Examples of translation of ER updates to NIAM updates.

5 Summary

We have integrated the conceptual data models for two common conceptual data modelling notions, ER and NIAM. This integrated model facilitates mapping dynamic data changes between the individual data models for each notation. These conceptual data models can form the basis for integrated CASE tool repositories, and we have implemented an I-CASE tool, NIAMER, which supports fully integrated NIAM and ER data modelling. Unlike other CASE tools, our data models are kept fully consistent, with any change to data in one model being translated into appropriate changes in the other model. Some changes can be automatically translated, while others require some degree of designer assistance. NIAMER always highlights such updates, and its default attempt at a translation, to assist designers in keeping their designs in each notation consistent.

We are currently integrating NIAMER and OOEER, to produce an I-CASE tool which supports EER, OOA/D and NIAM modelling, and relational schema and OO program construction. We are extending OOEER to support DFD and object-oriented functional diagram consistency. Keeping the data modelling parts of such diagrams consistent with ER and NIAM models is also being investigated. We are producing a metaCASE tool, incorporating our CoCoA modelling language and MViews framework. This will allow I-CASE tools such as NIAMER to be more easily specified, generated and integrated.

References

1. Campbell, L. and Halpin, T. Abstraction Techniques for Conceptual Schemas. In *Proceedings of the 5th Australasian Database Conference*, Global Publications Services, Christchurch, New Zealand, 17-18 January 1994, pp. 374-388.
2. Chen, P.P. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems 1*, 1 (1976), 9-36.
3. Falkenberg, E.D. and Oei, J.L.H.D.F.I.C.O.R.M. Meta Model Hierarchies from an Object-Role Modelling Perspective. In *.F.I.C.O.R.M. irst International Conference on Object-Role Modelling*, Halpin, T. and Meersman, R., University of Queensland, Brisbane, Australia, 4-6 July 1994, pp. 310-323.
4. Grundy, J.C. and Hosking, J.G. A framework for building visusal programming environments. In *Proceedings of the1993 IEEE Symposium on Visual Languages*, IEEE CS Press, 1993, pp. 220-224.
5. Grundy, J.C. and Venable, J.R. Providing Integrated Support for Multiple Development Notations. In *Proceedings of CAiSE'95*, LNCS 932, Springer-Verlag, Finland, June 1995, pp. 255-268.
6. Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Amor, R. Support for Collaborative, Integrated Software Development. In *Proceeding of the 7th Conference on Software Engineering Environments*, IEEE CS Press, Netherlands, April 5-7 1995, pp. 84-94.
7. Heym, M. and Österle, H. A Semantic Data Model for Methodology Engineering. In *Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering*, IEEE CS Press, Washington, D.C., 1992, pp. 142-155.
8. Meyers, S. Difficulties in Integrating Multiview Editing Environments. *IEEE Software 8*, 1 (January 1991), 49-57.
9. Nijssen, G.M. and Halpin, T.A. *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*, Prentice-Hall, Englewood Cliffs, NJ (1989).
10. Nuseibeh, B. and Finkelstein, A. ViewPoints: A Vehicle for Method and Tool Integration. In *Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering*, IEEE CS Press, Washington, D.C., 1992, pp. 50-61.
11. Ratcliffe, M., Wang, C., Gautier, R.J., and Whittle, B.R. Dora - a structure oriented environment generator. *IEE Software Engineering Journal 7*, 3 (1992), 184-190.
12. Reiss, S.P. Connecting Tools Using Message Passing in the Field Environment. *IEEE Software 7*, 7 (July 1990), 57-66.
13. Song, X. and Osterweil, L.J. A Process-Modeling Based Approach to Comparing and Integrating Software Design Methodologies. In *Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering*, IEEE CS Press, Washingon, D.C., 1992, pp. 225-229.
14. Venable, J.R. *CoCoA: A Conceptual Data Modelling Approach for Complex Problem Domains*, Ph.D. dissertation, Thomas J. Watson School of Engineering and Applied Science, State University of New York at Binghamton, 1993.
15. Verhijen, G.M.A. and van Bekkum, J. NIAM: An Information Analysis Method. In *Proceedings of the IFFIP WG 8.2. Working Conference on Comparative Review of Information Systems Design Methodologies*, North-Holland, New York, NY, 1982.
16. Wasserman, A.I. and Pircher, P.A. A Graphical, Extensible, Integrated Environment for Software Development. *SIGPLAN Notices 22*, 1 (January 1987), 131-142.